Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miloš Lukić

Primerjava algoritmov za rekonstrukcijo 3D modelov iz volumetričnih podatkov

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Matija Marolt

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomski nalogi preučite področje algoritmov za rekonstrukcijo površin iz 3D volumetričnih podatkov. Implementirajte pristop, ki ustvari 3D model na podlagi uteženih lokalnih implicitnih funkcij in Bloomenthalove poligonizacije ter ga optimizirajte za čim hitrejše izvajanje. Izvedite primerjavo rekonstruiranih površin z rezultati uveljavljene metode Marching cubes.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Miloš Lukić, z vpisno številko **63110202**, sem avtor diplomskega dela z naslovom:

Primerjava algoritmov za rekonstrukcijo 3D modelov iz volumetričnih podatkov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matije Marolta,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 9. septembra 2014 Podpis avtorja:

Zahvaljujem se družini in vsem znancem za podporo pri vseh letih študija. Zahvaljujem se doc. dr. Matiju Maroltu za mentorstvo in as. mag. Cirilu Bohaku za usmerjanje in strokovno pomoč pri izdelavi diplomske naloge.

Kazalo

Seznam uporabljenih kratic

kratica	**angleško**	**slovensko**
MPUI	multi-level partition	večnivojska delitev enote
	of unity implicits	z implicitnimi funkcijami
RBF	radial basis function	osnovna radialna funkcija
MC	marching cubes	metoda drsečih kock
MLS	marching least squares	metoda drsečih najmanjših kvadratov
3D	three-dimensional space	tridimenzionalni prostor

Povzetek

Cilj diplomske naloge je implementacija in primerjava algoritmov za rekonstrukcijo površin iz rezultatov segmentacije 3D volumetričnih podatkov. V že izdelan program za segmentacijo in prikazovanje modelov NeckVeins je implementiran pristop Multi-level partition of unity za generiranje uteženih lokalnih implicitnih funkcij. Iz omenjenih funkcij, ki aproksimirajo lokalno obnašanje vhodnih točk, je s postopkom izčrpne Bloomenthalove poligonizacije ustvarjen 3D model. Algoritem je optimiziran z uporabo k-d drevesa za iskanje najbližjih sosedov in s podporo za paralelno delovanje na večprocesorskih sistemih. Rezultat je javanski program, pri katerem lahko z različnimi parametri uravnavamo natančnost, hitrost in gladkost končnega 3D modela. Namesto dosedanje neposredne poligonizacije volumna s statično ločljivostjo se v algoritmu MPUI ustvarjena implicitna funkcija lahko rasterizira s poljubnim algoritmom za poligonizacijo implicitnih površin ali pa se neposredno vizualizira z različnimi metodami sledenja žarkom. Poligonizirani objekti so lahko v odvisnosti od parametrov določenih v aproksimaciji bolj gladki in lahko imajo višjo ločljivost kot neposredno poligonizirani volumni z metodo Marching cubes.

Ključne besede: Večnivojska delitev enote, implicitne funkcije, poligonizacija.

Abstract

The aim of the thesis is the implementation and comparison of algorithms for reconstructing 3D models from the segmented volumetric data. Multi-level partition of unity approach for generating weighted local implicit functions is implemented to existing program for visualising 3D models, NeckVeins. These functions are used to approximate local behaviour of input points. 3D model is then created using exhaustive Bloomenthal polygonization. The algorithm is optimized with the use of k-d trees for finding nearest neighbours and with support for parallel operation on multiprocessor systems. The result is a Java program with a variety of different parameters which control the accuracy, speed and smoothness of the final 3D model. Instead of a direct volume polygonization with static resolution, MPUI creates an implicit function that can be rasterized with any algorithm for implicit surface polygonization and is able to provide us with user-defined resolution of triangles. It can also be directly visualised with different ray tracing methods. Polygonized objects can be smoother and can have a higher resolution than those created with marching cubes method, which is limited with static resolution and direct polygonization.

Keywords: Multi-level partition of unity, implicit functions, polygonization.

Poglavje 1

Uvod

V zadnjih nekaj desetletjih se v znanosti in gospodarstvu vedno bolj uveljavlja tridimenzionalna (3D) računalniška vizualizacija podatkov, pridobljenih z različnimi metodami zajemanja informacij o obliki in površini objektov iz realnega življenja. Prav tako se v odvisnosti od panoge ali celo funkcije podatkov razlikujejo zahteve po kvaliteti določenih aspektov vizualizacije (hitrost, natančnost, preglednost in estetika rezultata). Pridobljene podatke je pogosto potrebno obdelati ali prestrukturirati za prikaz na računalniških sistemih, omenjeni postopki pa se lahko razlikujejo glede na vhodne podatke.

V medicini se uporabljajo 3D rentgenske tehnike za slikanje notranjih organov. Uveljavljene so predvsem tehnike rotacijske angiografije, računalniške tomografije (angl. computed tomography, CT) in slikanja z magnetno resonanco (angl. magnetic resonance imaging, MRI). V drugih panogah prevladuje 3D lasersko skeniranje, kjer so zajete le točke na površini objektov. Laserska skeniranja so tako uporabna pri zajemu geografskih podatkov in različnih objektov za uporabo na področjih kot so geodezija, arheologija, biologija in mnogih drugih. 3D podatki so se pojavili kot odgovor na vse večjo potrebo po globinskem prikazu objektov, ki so predmet preučevanja. S hitrim razvojem računalniške grafike se je pojavila tudi možnost intuitivnega prikaza velikega števila 3D podatkov. V tej diplomski nalogi bo poudarek na vizualizaciji 3D volumnov žil, ki lahko v primeru natančne in nazorne vi-

zualizacije učinkovito pomagajo pri diagnostiki nekaterih bolezni ožilja. Pri vizualizaciji je potrebno upoštevati šume in nizko ločljivost posnetkov, ki včasih ne zajamejo natančne površine slikanega objekta, ter ustvariti vizualizacijo, ki bo kljub pomanjkljivostim vhodnih podatkov čim bolj natančno predstavila slikani objekt.

Za vizualizacijo volumetričnih podatkov se uporabljajo poligonizacijske metode, katerih rezultat je površina sestavljena iz mnogokotnikov, ter metode neposrednega izrisovanja med katerimi je najbolj znana metoda metanje žarkov (angl. ray casting) [16] [15]. Prednost poligonizacijskih metod je hitrost izrisovanja končnega modela. V računalništvu so grafične kartice optimizirane za izrisovanje poligonov oziroma trikotnikov. To nam omogoča, da 3D površine, ki lahko v odvisnosti od zmogljivosti grafične kartice vsebujejo tudi več milijonov trikotnikov, izrisujemo v realnem času. Rezultat neposrednega izrisovanja z metodo metanja žarkov je bolj realistična slika objekta, vendar je simuliranje velikega števila žarkov lahko zelo počasno. V diplomski nalogi je poudarek predvsem na algoritmih katerih končni rezultat je poligoniziran objekt.

Delo bo potekalo na programu NeckVeins [5], ki ima implementirano funkcijo prikazovanja 3D objektov in poligonizacijo volumnov z algoritmom Marching cubes [20] [17]. Marching cubes je zaradi svoje preprostosti in hitrega izvajanja najbolj razširjena metoda za poligonizacijo volumetričnih podatkov, vendar je rezultat omejen na ločljivost volumna in strogo obliko segmentiranih struktur. Posledice so stopničaste površine in ostri robovi. Problem je delno rešljiv z glajenjem v stopnji segmentacije, vendar lahko s tem izgubimo precej podatkov (tanke žile v medicinskih volumnih). Zato je cilj implementirati pristop, ki bi brez posegov v segmentacijo ustvaril natančen model z gladko površino in poljubno visoko ločljivostjo. Probleme v veliki meri rešuje algoritem Multi-level partition of unity implicits (MPUI)[18]. Algoritem je vmesna stopnja med segmentiranimi rezultati in poligonizacijo. S kontroliranodelitvijo se iz oblaka točk aproksimirajo utežene implicitne funkcije, ki opisujejo površino objekta. Poleg MPUI sta implementirani tudi izčrpni po-

ligonizaciji implicitnih površin s tetraedri in kockami. Tako MPUI kot tudi poligonizacija sta optimizirana za večnitno delovanje. Opravljena je tudi primerjava vseh pristopov ter evalvacija natančnosti njihovih rekonstrukcij.

Poglavje 2

Pregled področja

Diplomsko delo sodi na področje računalniške grafike, ki se ukvarja z vizualnim prikazom tridimenzionalnih (3D) podatkov. 3D podatke je potrebno predstaviti v najbolj intuitivni obliki, pri vizualizaciji objektov iz realnega življenja je to 3D perspektivna projekcija. Za razumevanje problema vizualizacije medicinskih volumnov je potrebno razumeti vrste podatkov in zmožnosti njihovega prikaza na računalniku. Podatki pridobljeni iz 3D slikanja so v volumetrični obliki, te je možno vizualizirati z metodami neposrednega izrisovanja [16]. Hitrejši izris lahko dobimo s pretvorbo podatkov v primitive (rasterizacijo), ki jih lahko računalnik izriše (poligoni, trikotniki). Rasterizacija je lahko uporabljena neposredno na volumnu [17], v tem primeru je ločljivost primitivov omejena na ločljivost volumna. Ker je bolj kot natančnost vizualizacije volumna pomembna podobnost slikanemu objektu, je pogosto potrebna rekonstrukcija površine predstavljene v volumnu.

Osrednji algoritem diplomske naloge spada v skupino algoritmov, katerih namen je aproksimirati čim bolj natančno in gladko površino. Obstaja več pristopov k reševanju tega problema. Metoda omejenih elastičnih površinskih mrež (angl. constrained elastic surface nets, CESN) [10] je pristop, ki s pomočjo mreže povezanih vozlišč ustvari gladko površino. Težava tega pristopa je, da se lahko pri rekonstrukciji izgubijo tanke strukture [19]. Ker so nekatere sicer uspešne metode za rekonstrukcijo površin časovno zelo zah-

tevne (Poissonova rekonstrukcija [12], power crust [6]), so izločene iz podrobnejše obravnave. Bolj primerni so pristopi, kjer se iz volumna pridobi orientiran oblak točk, nad katerim se po območjih aproksimirajo implicitne funkcije. Metoda drsečih najmanjših kvadratov (angl. moving least squares - MLS) [14] za vsako vhodno točko v 3D domeni izračuna implicitno funkcijo po metodi najmanjših kvadratov in jo uteži glede na oddaljenost. Eden bolj znanih pristopov je tudi metoda radialnih osnovnih funkcij [8], ki z interpolacijo dosega dobre rezultate, vendar je kljub morebitnim aproksimacijskim pohitritvam izredno počasna [19].

Za implementacijo je izbran algoritem Multi-level partition of unity [18], saj poleg hitrejšega izvajanja in uravnoteženosti med natančnostjo in gladkostjo ponuja veliko prostora za izboljšave ter pohitritve. Algoritem, ki je bil prvotno namenjen rekonstrukciji podatkov pridobljenih s tehniko 3D laserskega skeniranja, se je izkazal uporaben tudi pri rekonstrukciji volumnov žil [19]. Koncept MPUI je zelo podoben pristopu MLS, vendar vsebuje dve ključni novosti, implicitne funkcije definirane na več nivojih (posledica delitve z osmiškim drevesom) in utežne funkcije, ki delujejo kot delitev enote (angl. partition of unity).

Poglavje 3

Algoritmi za rekonstrukcijo površin

Generiranje poligoniziranega objekta z algoritmom MPUI poteka v več korakih (slika 3.1). Vhodna podatka sta segmentirana 3D matrika, kjer vsaka vrednost predstavlja eno enoto imenovano voksel, in prag, ki določa kateri voksli predstavljajo objekt in kateri prazen prostor. Ker MPUI za vhodni podatek potrebuje oblak orientiranih točk, je v prvem koraku potrebno iz rezultata segmentacije pridobiti točke v 3D prostoru ter pripadajoče normale.

Dobljene točke se v drugi stopnji proporcionalno skalira tako, da so omejene s prostorom v obliki kocke (v nadaljevanju "osnovna celica"), ki ima stranico dolgo eno enoto in vse stranice poravnane s koordinatnimi osmi v evklidskem prostoru. Priporočeno je, da se skalirane točke pomakne v središče osnovne celice, zaradi statične delitve prostora v naslednji stopnji (bolj kot je porazdelitev točk enakomerna, manj je praznih celic in bolj uravnoteženo je drevo). V tej stopnji je priporočena ustvaritev k-d drevesa za iskanje najbližjih točk.

Tretji korak obsega algoritem MPUI in ustvaritev množice uteženih implicitnih funkcij. Dobljene implicitne funkcije omogočajo preslikavo poljubnih 3D točk v realno vrednost:

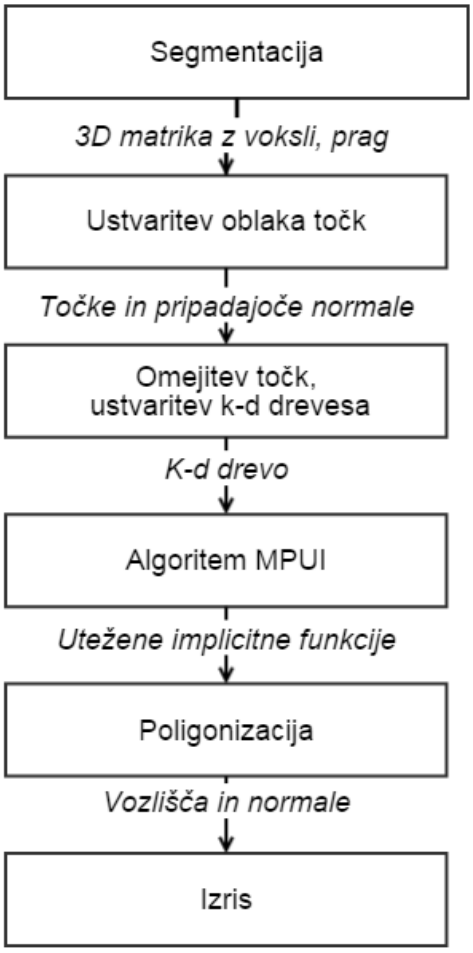
$$f(x, y, z) = 0,$$
$$\mathbb{R}^3 \to \mathbb{R}$$

Funkcije so na lokalni ravni aproksimirane tako, da imajo vrednost 0 na površini objekta, v notranjosti objekta negativne vrednosti ter pozitivne vrednosti zunaj objekta. Ker so implicitne funkcije aproksimirane lokalno, je njihov vpliv zunaj aproksimacijskega območja potrebno omejiti. Iz lokalnosti aproksimacij poleg omejitve vpliva izvira tudi problem nesklenjenosti funkcij. Lokalne funkcije v veliki večini primerov aproksimirajo točke, ki predstavljajo le eno površino. Zato je smiselna uporaba metode, ki omogoča zlivanje sosednjih funkcij. Rešitev je uporaba utežnih funkcij, ki delujejo na podoben način kot radialne funkcije (angl. radial functions). Ključna lastnost teh funkcij je odvisnost vrednosti funkcije od oddaljenosti od izhodišča (pri MPUI se izhodišča nahajajo v središčih prostorov, kjer se izvaja lokalna aproksimacija). Za uspešno rekonstrukcijo je izredno pomembna delitev enote (angl. partition of unity). Za poljubno točko v osnovni celici mora veljati da je vsota pomožnih funkcij (iz katerih so izpeljane uteži) enaka ena. V naslednjih enačbah je prikazana lastnost delitve enote v odvisnosti od uteži w_i in pomožnih funkcij φ:

$$f(x, y, z) = \frac{\sum_i w_i(x, y, z) Q_i(x, y, z)}{\sum_i w_i(x, y, z)}$$
$$= \sum_i \varphi_i(x, y, z) Q_i(x, y, z) \tag{3.1}$$
$$\sum_i \varphi_i(x, y, z) = 1 \tag{3.2}$$

Dobljeno množico uteženih implicitnih funkcij lahko v zadnjem koraku vizualiziramo s poligonizacijo, ali pa s Hartovo metodo včrtavanja sfer (angl. sphere tracing) [11]. Za potrebe te naloge je bila implementirana poligonizacija, katere koncept je podoben Bloomenthalovi [7]. Vendar je zaradi

prisotnosti ločenih konstruktov v volumnu uporabljena izčrpna poligonizacija namesto metode sledenja površini. Rezultat poligonizacije so vozlišča in normale, ki v trojicah določajo trikotnike za izris. Vozlišča so v geometriji definirana kot presečišča geometrijskih oblik. V našem primeru definirajo presečišča oziroma oglišča trikotnikov.



Slika 3.1: Diagram cevovoda za rekonstrukcijo volumna z algoritmom MPUI. V pravokotnikih so predstavljeni postopki, v povezavah pa vhodni oziroma izhodni podatki.

3.1 Ustvaritev oblaka točk

Oblak točk je definiran kot množica točk v evklidskem prostoru. Te imajo za vsako dimenzijo podano eno realno vrednost. Največkrat se uporablja za definicijo točk na površini objekta, zato je uporaben pri 3D laserskih skeniranjih. Oblak točk je lahko upodobljen neposredno (slika 3.3), vendar se zaradi neintuitivnosti globinskega pogleda in nesklenjenosti ne uporablja za vizualizacijo medicinskih volumnov. Za algoritem MPUI potrebujemo poleg informacije o lokaciji tudi informacijo o orientaciji točk. Informacija o orientiranosti nam pomaga predvsem pri odpravljanju topoloških dvoumnosti, ki se pojavijo pri kompleksnejših strukturah. Orientiranost točk podamo v obliki normal, ki so izračunane za vsako posamezno točko. Normale so normirani vektorji (dolžina normiranega vektorja je vedno enaka 1), ki določajo v katero smer je obrnjena površina.

3.1.1 Implementacija

Pretvorbe rezultatov segmentacije v oblak točk zahteva preprosto implementacijo. Za vsak voksel v 3D mreži se v odvisnosti od sosednjih vokslov postavi točke in izračuna njihove normale. Točke so postavljene le v vokslih, ki ne dosegajo določenega praga, a se stikajo z voksli, ki ga presegajo in tako predstavljajo mejne voksle oziroma površino objekta. Vrednosti vokslov v volumnu so predstavljene kot pozitivna števila. Odvisno od formata slikanja so lahko osem ali šestnajst bitna. Prag je število med 0 in $2^n - 1$, kjer je n število bitov, uporabljenih za predstavitev vrednosti posameznih vokslov. Voksli, ki presegajo ta prag, določajo notranjost objekta, ostali pa določajo prazen prostor.

Glede na število in postavitev sosedov, ki presegajo prag, obstajajo različne konfiguracije za postavljanje in orientiranje točk. Za vsak voksel v mreži se preveri ali presega prag. Če vrednost voksla ne presega praga, se v odvisnosti od rezultata preveri še šest vokslov, ki imajo s trenutno izbranim deljene ploskve (definiramo jih kot sosednje voksle). Normale posameznih točk

izračunamo kot vsoto vseh nasprotnih vrednosti relativnih indeksov vokslov (položaj glede na trenutno izbrani voksel), ki se stikajo oziroma neposredno določajo točko. V večini primerov točko neposredno določajo vsi voksli, ki presegajo prag. Izjema so le konfiguracije z voksli, ki si ležijo nasproti in nimajo tranzitivnih povezav preko robov ter njihovi inverzi.

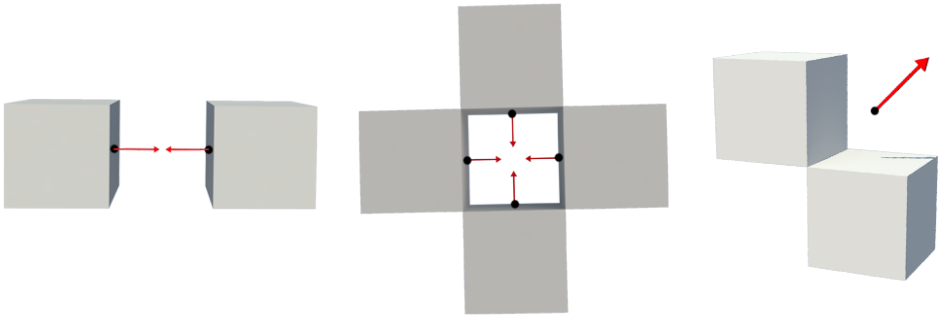
V trenutni prostor ne postavimo točk, če:

1. Vrednost voksla presega prag.
2. Vrednost voksla ne presega praga in prav tako ga ne presegajo vrednosti sosednjih vokslov

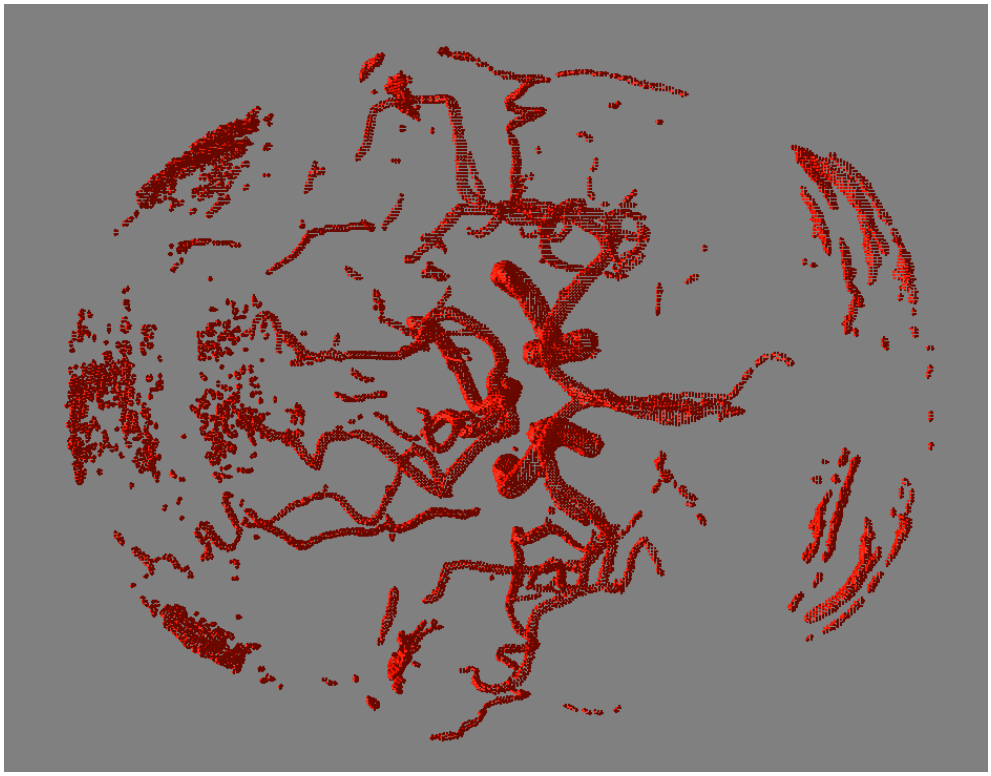
V vseh ostalih primerih je v prostor postavljena vsaj ena točka (slika 3.2). Te konfiguracije so sledeče:

1. Če prag presegata dva sosednja voksla, ki si ležita nasproti, se na središče vsake ploskve postavi po ena točka in se shranita njuni normali.
2. Če prag presegajo štirje voksli, ki ležijo v isti ravnini (nasprotje zgornji konfiguraciji), se prav tako postavi točke na središča vseh štirih vokslov in se shranijo njihove normale.
3. V vseh ostalih primerih se točka postavi v center trenutnega voksla, normala se shrani kot vsota normal stičnih ploskev sosednjih vokslov, katerih vrednost prestopa prag.

Izgradnja oblaka točk ima zaradi lokalne narave obravnavanja posameznih vokslov visoko zmožnost paralelizacije. Vendar je tudi zaporedna implementacija v primerjavi z drugimi koraki postopka rekonstrukcije relativno hitra. Zato paralelizacija tega koraki ni bila implementirana.



Slika 3.2: Postavitve točk v primeru dveh nasprotnih vokslov, štirih vokslov v isti ravnini ter poljubnega števila vokslov z deljenimi robovi.



Slika 3.3: Oblak točk ustvarjen iz volumna pridobljenega s postopkom 3D angiografije človeške glave

3.2 K-d drevo

Pred izgradnjo k-d drevesa je potrebno vse elemente oblaka točk omejiti v 3D celico z enakimi stranicami dolžine ena. Točke dobljene pri ustvaritvi oblaka točk se v tej stopnji proporcionalno skalira glede na dimenzije začetnega volumna matrike, tako da je najdaljša dimenzija matrike dolga eno enoto. Omejevanje položaja točk je bistveno predvsem za parametre, ki določajo delovanje algoritma MPUI. Ti parametri so lahko v primeru statično omejenega prostora oblaka točk konstantni in so posledično konstantni tudi rezultati pri različnih vhodnih podatkih. Ko so vse točke transformirane v celico, se prične izgradnja k-d drevesa.

K-d drevo (okrajšava za k-dimenzionalno drevo) je drevesna podatkovna struktura, ki organizira podatke v k-dimenzionalnem prostoru. Pri implementaciji algoritma MPUI je k-d drevo uporabno za iskanje najbližjih sosedov. Brez k-d drevesa je iskanje naivno in se je potrebno sprehoditi čez vse točke, da bi našli tiste, ki se nahajajo v določenem območju. Pri iskanju točk v določenem območju z uporabo k-d drevesa lahko zavržemo vozlišča (tudi celotna poddrevesa), v primeru da vrednost dimenzije trenutnega vozlišča ni vsebovana v iskalnem območju.

Podatki so binarno razdeljeni glede na lokacijo v določeni dimenziji. Vsako vozlišče v k-d drevesu predstavlja eno točko v prostoru. Ideja je, da se na vsaki globini drevo razdeli na podatke, katerih vrednost je v trenutno izbrani dimenziji manjša od mediane in na tiste, kjer je ta vrednost večja. Podatke z vrednostjo manjšo od mediane, se postavi v nadaljnjo obravnavo v levo hčerinsko vozlišče, večje pa v desno. V naslednji stopnji se v obeh hčerinskih vozliščih podatki ločijo glede na mediano v drugi dimenziji. Postopek se nadaljuje dokler ne zmanjka vozlišč. Dimenzije po katerih ločimo podatke se z globino drevesa izmenjujejo v istem vrstnem redu in z enako zastopanostjo. Postopek grajenja drevesa ima časovno zahtevnost $O(nlogn)$.

3.2.1 Implementacija

Izgradnja drevesa je implementirana rekurzivno, kot vhodni podatek so podana tri polja (za vsako dimenzijo po eno polje) Vsako polje je sortirano po velikosti glede na eno izmed dimenzij. Poleg sortiranih polj, ki omogočajo lažje iskanje mediane, je potreben tudi podatek o globini drevesa, preko katerega se izračuna po kateri dimenziji se podatke loči. Za lažje razumevanja algoritma je podana psevdokoda:

Algoritem 1 Ustvaritev k-d drevesa

```
1:  procedure ustvariVozlisce(globina, sortiraneTocke)
2:      dimenzija ← globina  mod steviloDimenzij
3:      steviloTock ← dolzina(steviloDimenzij[0])
4:      if steviloTock = 0 then
5:          return NULL
6:      if steviloTock = 1 then
7:          return novoVozlisce(sortiraneTocke[0][0], NULL, NULL)
8:
9:      mediana ← poisciMediano(sortiraneTocke[dimenzija])
10:     leviSeznam ← seznamVozliscPoDimenzijah().
11:     desniSeznam ← seznamVozliscPoDimenzijah().
12:
13:     for d in 0...steviloDimenzij do
14:         for t in 0...sortiraneTocke[d] do
15:             if t < mediana then
16:                 leviSeznam[d].dodaj(t)
17:             else if indeks(t)! = indeks(mediana) then
18:                 desniSeznam[d].dodaj(t)
19:
20:     leviOtrok = ustvariVozlisce(globina+1, leviSeznam)
21:     desniOtrok = ustvariVozlisce(globina+1, desniSeznam)
22:     return novoVozlisce(mediana, leviOtrok, desniOtrok)
```

Iskanje točk, ki ležijo v nekem območju zahteva preprosto implementacijo. Najprej se preveri, ali je začetno (korensko) vozlišče v prvi dimenziji večje, manjše in ali seka to območje. V primeru da je večje, pregledamo levega otroka po naslednji dimenziji, če je manjše pregledamo desnega in če seka to območje pregledamo oba. V zadnjem primeru se preveri tudi ali točka v tem vozlišču leži v območju in če je temu tako, se jo doda v rezultat.

3.3 Izgradnja osmiškega drevesa

Oblak orientiranih točk, ki je omejen s celico v obliki kocke s stranico dolgo eno enoto in središčem v koordinatnem izhodišču, je pripravljen za aproksimacijo implicitnih funkcij z algoritmom MPUI. Koncept algoritma je preprost, po metodi najmanjših kvadratov se izračuna aproksimacija kvadratne funkcije iz točk vsebovanih v prostoru. Če je izračunana napaka aproksimirane funkcije večja od uporabniško definirane, se z uporabo osmiškega drevesa prostor razdeli na več podprostorov in se v vsakem izmed njih funkcije znova aproksimira, tokrat na podlagi manjšega nabora točk. Ločene funkcije se nato zlije skupaj, kot je opisano v poglavju 3.6.

Osmiško drevo (angl. octree) je drevesna struktura, ki za razliko od k-d drevesa ne uporablja binarne, temveč osmiško razdelitev. Poleg te razlike je pomembna tudi statična delitev osmiškega drevesa. To pomeni, da ne upošteva porazdelitve točk v prostoru, temveč se vedno deli na enak način. Osmiško drevo se največkrat uporablja pri razdelitvi 3D prostora na podprostore. Je analogno štiriškemu drevesu (angl. quadtree) za delitev dvodimenzionalnega prostora.

V kontekstu algoritma MPUI je delitev osmiškega drevesa definirana kot delitev osnovne celice na osem enako velikih podprostorov. Prednost, ki jo prinese uporaba osmiškega drevesa, je prilagodljivost globine kompleksnosti objekta, ki ga želimo vizualizirati. Prilagodljivost tako omogoči aproksimacijo preprostih struktur z zelo majhnim številom delitev, kar poleg hitrejšega izračuna globalne implicitne funkcije vodi tudi k hitrejši poligonizaciji.

Vsaka celica ima podporni polmer, katerega dolžina je odvisna od dolžine stranice ali prostorske diagonale celice (v tej implementaciji je izbrana odvisnost od prostorske diagonale). Točke, katerih oddaljenost od središča celice je manjša od dolžine podpornega polmera, so uporabljene za aproksimacijo lokalne implicitne funkcije v celici. Podporni polmer določimo s parametrom α [18]:

$$R = \alpha\sqrt{3} \tag{3.3}$$

Večji kot je podporni polmer, večji bo deljeni prostor funkcij (presek krogel, ki jih določajo podporni polmeri celic) in več funkcij bo določalo globalno vrednost. Večji podporni polmer pripomore k bolj gladkemu rezultatu aproksimacije, v zameno za daljši daljši čas izvajanja programa.

Za aproksimacijo v vsaki celici se uporabijo točke znotraj podpornega polmera. Ker se uporablja statična delitev prostora, se lahko zgodi da nekatere celice vsebujejo premalo točk ali pa so prazne. Za obravnavo takih primerov je uporabljen t.i. pomožni polmer R', katerega vloga je podobna vlogi podpornega polmera. Razlika je ta, da pomožni polmer pride v poštev takrat, ko celica z definiranim podpornim polmerom vsebuje premalo točk. Pomožni polmer se iterativno povečuje, dokler ne vsebuje zadostnega števila točk za aproksimacijo. Povečevanje pomožnega polmera je določeno s konstanto λ. Pri prvem pregledu števila vsebovanih točk se definira:

$$R' = R \tag{3.4}$$

V primeru premajhnega števila vsebovanih točk, se pomožni polmer iterativno povečuje v odvisnosti od podpornega polmera:

$$R' = R' + \lambda R \tag{3.5}$$

Ko so točke, ki so pod vplivom celice določene, se izvede aproksimacija po metodi najmanjših kvadratov (poglavji 3.4 in 3.5). Delitev osmiškega drevesa je nato določena z enačbo [9] [18]:

$$\max_{|p_i - c| < R} \frac{|Q(p_i)|}{|\bigtriangledown Q(p_i)|} < \varepsilon_0 \tag{3.6}$$

kjer je ε_0 uporabniško določen parameter, ki predstavlja najmanjšo možno vrednost, pri kateri se drevo deli še naprej. Točke p_i so vse točke, ki so zajete s pomožnim polmerom (ki je lahko večji ali enak podpornemu polmeru). V primeru da je celica s podpornim radijem R prazna, nobena točka ne zadošča pogoju za iskanje maksimalne vrednosti v enačbi (3.6), kar povzroči, da se celica ne deli naprej.

3.4 Kvadratna implicitna funkcija

Problem, ki ga je v tej stopnji potrebno rešiti, je aproksimacija kvadratne implicitne funkcije iz nabora n točk $P = \{p_1, p_2, p_3...p_n\}$. Kvadratna implicitna funkcija je oblike:

$$\begin{aligned} Q(x,y,z) = Ax^2 + By^2 + Cz^2 + 2Dxy + 2Eyz \\ +2Fzx + 2Gx + 2Hy + 2Iz + J = 0 \end{aligned} \tag{3.7}$$

Za definicijo funkcije je potrebno poiskati vseh deset koeficientov (A, B, C, D, E, F, G, H, I in J). Spremenljivke x, y in z določajo koordinate. Funkcijo se lahko z danimi točkami takoj aproksimira po metodi najmanjših kvadratov, vendar je ključnega pomena tudi njihova orientiranost. Zato se uporablja t.i. pomožne točke.

Za najboljši rezultat je izbranih devet pomožnih točk. Izbrana so vsa oglišča celice ter njeno središče. Za vsako izmed pomožnih točk q se poišče k najbližjih točk (poimenujemo jih kontrolne točke), s katerimi se preveri zanesljivost pomožnih točk. Pomožna točka je zanesljiva takrat, ko je odstopanje kotov med vektorjem razdalje kontrolne in vsake izmed pomožnih točk ter pripadajočim normalnim vektorjem konsistentno. Ta pogoj se preveri z izračunom skalarnega produkta med vektorjem razdalje pomožne točke in vsake izmed kontrolnih točk ter vektorjem normale, ki pripada posamezni točki. Če so vsi skalarni produkti enakega predznaka je pomožna točka zanesljiva, sicer se zavrže. Za vsako pomožno točko, ki ni bila zavržena, se izračuna aritmetično povprečje skalarnih produktov, kot je prikazano na

enačbi [18]:

$$d_i = \frac{1}{6} \sum_i n_i \cdot (q - p_i) \tag{3.8}$$

Po preverjanju pomožnih točk z aritmetičnim povprečjem skalarnih produktov se lahko izračuna aproksimacija po metodi najmanjših kvadratov. Vse točke iz nabora P se prestavi v lokalni koordinatni sistem, ki ima središče v uteženem povprečju vsote vseh točk in osi vzporedne z osmi globalnega koordinatnega sistema.

Vsaka izmed točk se nato vstavi v enačbo splošne kvadratne funkcije (3.7). Vse dobljene funkcije se uteži glede na oddaljenost vstavljene točke od središča celice in se jih nato kvadrira. Rezultat je n matrik velikosti 10x10, kjer je n število točk v naboru P. Izračuna se vsota vseh dobljenih matrik. Nato se v enačbo splošne kvadratne funkcije vstavijo še zanesljive kontrolne točke. Vsota kvadratov dobljenih funkcij se sešteje s prej dobljeno matriko. Postopek je opisan z enačbo:

$$A = \sum_i^n \varphi(p_i) Q(p_i)^2 + \sum_i^m \frac{1}{m} Q(q_i)^2 \tag{3.9}$$

Podobno se izračuna tudi vektor b, ki bo uporabljen pri vzvratni substituciji:

$$b = \sum_i^m \frac{1}{m} Q(q_i) d_i. \tag{3.10}$$

Za iskanje minimuma $||Ax - b||$ je na matriki A uporabljen singularni razcep (angl. singular value decomposition, SVD). Če je matrika A dimenzij $n \times m$, so rezultat singularnega razcepa matrika U velikosti $n \times n$, matrika Σ velikosti $n \times m$ in matrika V velikosti $m \times m$. Matriki U in V sta ortogonalni, Σ pa je diagonalna matrika, ki vsebuje lastne vrednosti. Vektorji $U = [u_1, u_2, \ldots u_n]$ so levi singularni vektorji, $V = [v_1, v_2, ...v_n]$ pa desni singularni vektorji. Razcep je opisan z naslednjo enačbo:

$$U\Sigma V^T = A \tag{3.11}$$

Matrike dobljene s singularnim razcepom se uporabi pri vzvratni substituciji (enačba 3.12). $\{\sigma_1, \sigma_2, ...\sigma_m\}$ so lastne vrednosti dobljene iz diagonale matrike Σ.

$$x = \sum_{i=1}^{n} \frac{u_i^T b}{\sigma_i} v_i \tag{3.12}$$

Vektor x je dolžine 10 in vsebuje aproksimirane parametre za splošno kvadratno funkcijo (3.7). Funkcija je aproksimirana v lokalnih koordinatah, zato je potreben premik v globalni koordinatni sistem. Spodnja enačba prikazuje premik funkcije za vektor $v = [x_t, y_t, z_t]$ (preslikava v globalni koordinatni sistem):

$$\begin{aligned} G' &= G - Dy_t - Fz_t - 2Ax_t \\ H' &= H - Ez_t - Dx_t - 2By_t \\ I' &= I - Fx_t - Ey_t - 2Cz_t \\ J' &= J - Ix_t - Hy_t - Gz_t \\ &+ Dx_t y_t + Ey_t z_t + Fz_t x_t \\ &+ Ax_t^2 + By_t^2 + Cz_t^2 \end{aligned} \tag{3.13}$$

Dobljeni parametri G', H', I' in J', zamenjajo parametre G, H, I in J v splošni kvadratni funkciji (3.7).

3.5 Bivariatna polinomska funkcija

Bivariatna polinomska funkcija je namenjena aproksimaciji nabora točk $P = \{p_1, p_2, ...p_n\}$, kjer je prisotno majhno odstopanje med koti normal. Pogoj je izpolnjen, če je maksimalni kot Θ med uteženim povprečjem normal in vsako normalo posamezne točke manjši od 90 stopinj. Povprečno normalo

se izračuna kot normirano uteženo vsoto vseh normal iz nabora točk. Ker je orientiranost površine znana, je aproksimacija preprostejša kot pri splošni kvadratni funkciji. Uporabljena je kvadratna bivariatna polinomska funkcija. Za njeno aproksimacijo je potrebno aproksimirati vrednosti šestih parametrov (A, B, C, D, E in F). Enačba bivariatne funkcije je:

$$f(x, y) = Ax^2 + By^2 + Cxy + Dx + Ey + F \tag{3.14}$$

Aproksimacija poteka podobno kot pri splošni kvadratni funkciji, vendar obstaja nekaj bistvenih razlik. Pri bivariatni polinomski funkciji se definira nov lokalni koordinatni sistem, katerega osi (v večini primerov) niso poravnane z osmi globalnega koordinatnega sistema. Globalne koordinate točk (x, y in z) se preslikajo v lokalne (i, j in k). Lokalna koordinatna os k je enaka uteženemu povprečju normal, ostali dve koordinatni osi pa sta ortogonalni osi k (skalarni produkt poljubnih parov osi je enak 0). Koordinatno izhodišče je postavljeno v aritmetično povprečje uteženih točk.

Za aproksimacijo se uporabi metoda najmanjših kvadratov. Iz vsote kvadratov uteženih funkcij, v katere se vstavi lokalne koordinate vsake izmed točk, se dobi kvadratno matriko A, velikosti 6x6. Spodnja funkcija opisuje izračun matrike A:

$$A = \sum_{i}^{n} w_i f(p_i)^2. \tag{3.15}$$

Vektor b, ki je uporabljen za vzvratno substitucijo, se dobi s pomočjo vrednosti lokalne koordinate k:

$$b = \sum_{i}^{n} w_i k_i f(p_i). \tag{3.16}$$

Postopek za minimizacijo $||Ax-b||$ je enak kot pri splošni kvadratni funkciji, opisan je v enačbah (3.11) in (3.12). Rezultat je vektor x, ki vsebuje vseh

6 aproksimiranih neznank iz enačbe (3.14). Zaradi potrebe po standardiziranosti lokalnih funkcij se dobljeno bivariatno funkcijo pretvori v kvadratno implicitno funkcijo in lokalne koordinate prevede v koordinate, ki so poravnane z globalnimi ($A- > A'$). S projekcijami ortonormirane baze vektorjev $\{i, j, k\}$, ki določa lokalni koordinatni sistem, se dobi vse potrebne parametre za splošno kvadratno funkcijo:

$$\begin{aligned}
A' &= -Ai_x^2 - Bj_x^2 - Ci_xj_x \\
B' &= -Ai_y^2 - Bj_y^2 - Ci_yj_y \\
C' &= -Ai_z^2 - Bj_z^2 - Ci_zj_z \\
D' &= -(2(Ai_xi_y + Bj_xj_y) + C(i_xj_y + i_yj_x)) \\
E' &= -(2(Ai_yi_z + Bj_yj_z) + C(i_yj_z + i_zj_y)) \\
F' &= -(2(Ai_zi_x + Bj_zj_x) + C(i_zj_x + i_xj_z)) \\
G' &= k_x - Di_x - Ej_x \\
H' &= k_y - Di_y - Ej_y \\
I' &= k_z - Di_z - Ej_z \\
J' &= F
\end{aligned} \tag{3.17}$$

Dobljeni parametri so enakovredni parametrom dobljenim z vzvratno substitucijo pri splošni kvadratni funkciji (3.12). Funkcijo se nato premakne v koordinatno izhodišče. Tudi ta postopek je opisan pri aproksimaciji s splošno kvadratno funkcijo (3.13).

3.6 Utežne funkcije

Utežne funkcije se pri MPUI algoritmu uporablja za zlivanje lokalnih implicitnih funkcij. Cilj utežnih funkcij je ustvariti delitev enote na območju osnovne celice. To pomeni, da je seštevek prispevkov vseh uteži k posameznim lokalnim funkcijam v poljubni točki 3D prostora osnovne celice vedno

enak ena. Prispevek posamezne funkcije opisuje naslednja enačba:

$$f_i(x) = \frac{w_i(p_i, c_i, r_i)Q_i(p_i)}{\sum_i w_i(p_i, c_i, r_i)}. \tag{3.18}$$

Utež, ki pripada posamezni funkciji je odvisna od položaja točke p_i, središča celice funkcije c_i in podpornega polmera funkcije r_i. Pri računanju z utežmi je uporabljen podporni polmer namesto pomožnega, ki se uporablja le za namene aproksimacije (v nasprotnem primeru bi pri praznih celicah povzročal nenatančne rezultate).

Za namene aproksimacije se uporablja funkcija osnovnih zlepkov (angl. basis spline ali B-spline) [18]. Gladkost končne površine je v veliki meri odvisna od uporabljene utežne funkcije. V naslednji enačbi je definiran odnos med danimi parametri:

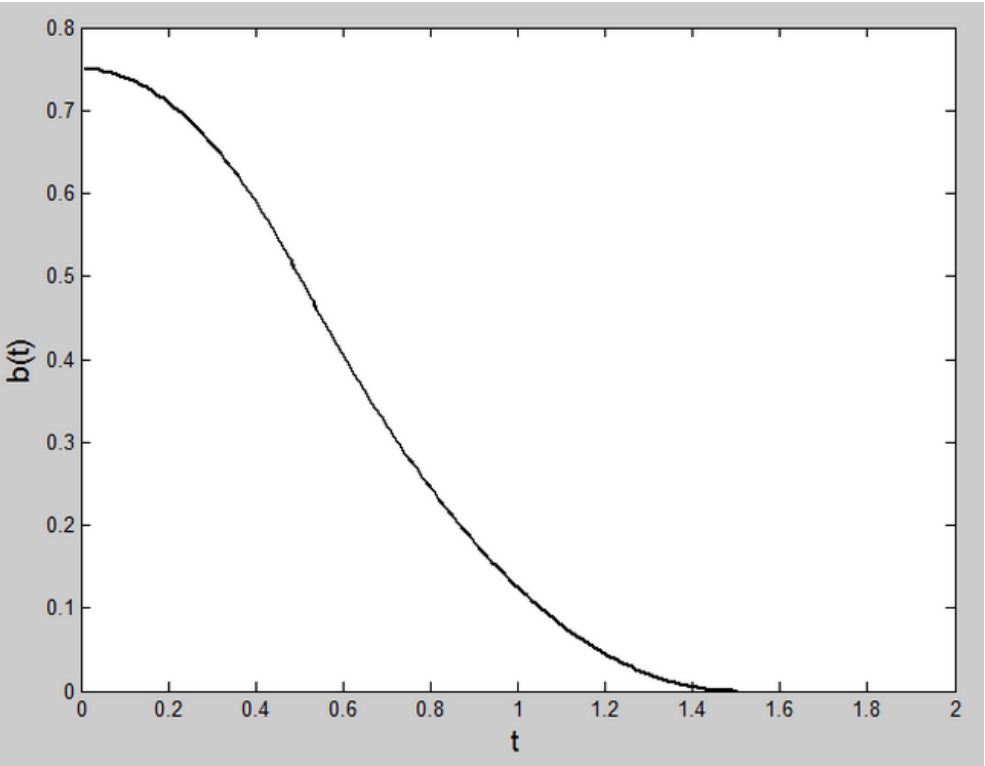
$$w_i(p_i, c_i, r_i) = b(\frac{k|p_i - c_i|}{r_i}), \tag{3.19}$$

Izračuna se razdalja med dano točko in središčem celice ter razmerje dobljene razdalje s podpornim polmerom celice. Večja ko je razdalja, večje je razmerje, ki se nato kot parameter poda v funkcijo osnovnih zlepkov. Poleg omenjenega razmerja je podan še parameter k, ki je pomemben za sovpadanje parametra s področjem definiranosti funkcije osnovnega zlepka. Posredovani parameter zato vsebuje le vrednosti med 0 (če je točka na središču celice) in k (če je točka od središča oddaljena za dolžino podpornega polmera).

Za izračun vrednosti glede na posredovani parameter se za potrebe te naloge uporablja kvadratna funkcija osnovnih zlepkov (po vzoru na [18]). Funkcija je definirana kot:

$$b(t) = \begin{cases} -t^2 + 0.75 & : 0 \leqslant t \leqslant 0.5 \\ \dfrac{(1.5 - t)^2}{2} & : 0.5 < t \leqslant 1.5 \end{cases} \tag{3.20}$$

Cilj dane funkcije je preslikati majhne vrednosti (ki so blizu ničle) v visoke, saj morajo točke ali funkcije, ki so bližje središču celice, imeti večji vpliv. Graf funkcije prikazuje slika 3.4.



Slika 3.4: Preslikava vrednosti parametra t s funkcijo osnovnih zlepkov (3.20).

3.7 Implementacija in paralelizacija MPUI

Algoritem MPUI je implementiran v programu NeckVeins, ki obsega branje volumnov in njihovih informacij v formatu .mhd, izrisovanje modelov iz datotek formata .obj, segmentacijo in poligonizacijo volumnov, grafični vmesnik ter ostale funkcionalnosti. NeckVeins uporablja knjižnico LWJGL (angl. The lightweight java game library), ki omogoča uporabo knjižnic OpenGL (angl. Open graphics library), OpenCL (angl. Open computing language) in OpenAL (angl. Open audio library). Izris poligoniziranih površin omogoča OpenGL, ki deluje na različnih platformah in lahko izkorišča strojno opremo, kot je grafična kartica, za pospešen izris.

Poleg knjižnice LWJGL je uporabljena tudi knjižnica EJML (angl. Efficient java matrix library), ki omogoča uporabnikom izvajanje različnih operacij nad matrikami. Uporabljena je za računanje singularnih razcepov pri aproksimaciji. V primerjavi z ostalimi knjižnicami, ki omogočajo SVD, je knjižnica EJML pri manjših matrikah (velikosti 6 in 10) najbolj učinkovita [4]. EJML omogoča singularni razcep samo na matrikah, ki imajo elemente shranjene v dvojni natančnosti. Je javanska knjižnica, zato je njena vključitev v projekt preprosta. Prav tako ima pri mnogih operacijah na matrikah boljše rezultate kot večina domorodnih (angl. native) knjižnic (npr. JBLAS, Java basic linear algebra subprograms, ovojna knjižnica za fortranove rutine vektorskih in matričnih operacij).

Zaradi lokalnih aproksimacij ima MPUI visoko zmožnost paralelizacije. Manjše napake in večji podporni polmeri celic povzročijo globoke delitve drevesa in posledično več aproksimacij, kar lahko bistveno poveča čas izvajanja. Paralelizacija na grafični kartici je zaradi kompleksnih drevesnih struktur (osmiško drevo in k-d drevo), mnogih pogojnih ukazov ter kompleksnih postopkov (SVD) težavna in ne zagotavlja pohitritve ali celo uspešnosti izvajanja. Za potrebe te naloge je bila implementirana statična procesorska paralelizacija z omejitvijo računanja na največ osem paralelnih enot. Osnovna celica se statično razdeli na osem delov. Vsaka izmed niti nato obravnava svoj del kot osnovno celico. Pohitritev paralelizacije je odvisna od poraz-

deljenosti vhodnih podatkov, zato je izrednega pomena postavitev središča vhodnih podatkov v središče osnovne celice. Če je prisotnih osem računskih enot, ki lahko paralelno izvajajo operacije v enojni in dvojni natančnosti, je v najboljšem primeru lahko pohitritev osemkratna. To je več kot dovolj za večino osebnih računalnikov, ki imajo ponavadi dve ali štiri jedra in omogočajo paralelno izvajanje prej omenjenih operacij.

Za paralelizacijo je bil izbran javanski vmesnik Callable, ki omogoča večnitno izvajanje programske kode. Je različica vmesnika Runnable, vendar ima zmožnost vračanja rezultata in sprožanja preverjenih izjem [1].

Poglavje 4

Algoritmi za poligonizacijo implicitnih površin

Za ustvaritev interaktivnega prikaza 3D modela so implicitne funkcije rasterizirane z uporabo trikotnikov. Koncept vseh poligonizacij uporabljenih v tej nalogi je enak. Izbrana je poljubno majhna celica, ki se odvisno od algoritma premika po poligonizacijskem prostoru. Pri vsakem premiku se v vsakem izmed oglišč celice preverijo vrednosti implicitne funkcije. Celica seka površino modela, če se vrednosti v ogliščih ne ujemajo v predznaku. V tem primeru se glede na algoritem ustrezno postavijo trikotniki. Pri postavitvi trikotnikov je pomembna bisekcija, s katero se oglišča trikotnikov postavi čim bližje vrednosti nič, ki določa površino. Težava tega pristopa je poligonizacija tankih površin, saj je za to potrebno uporabiti manjšo velikost poligonizacijske celice, kar povzroči daljši čas računanja in lahko ustvari veliko število trikotnikov. Poligonizacija celice je glede na postavitev trikotnikov implementirana na dva načina, s tetraedri in kockami.

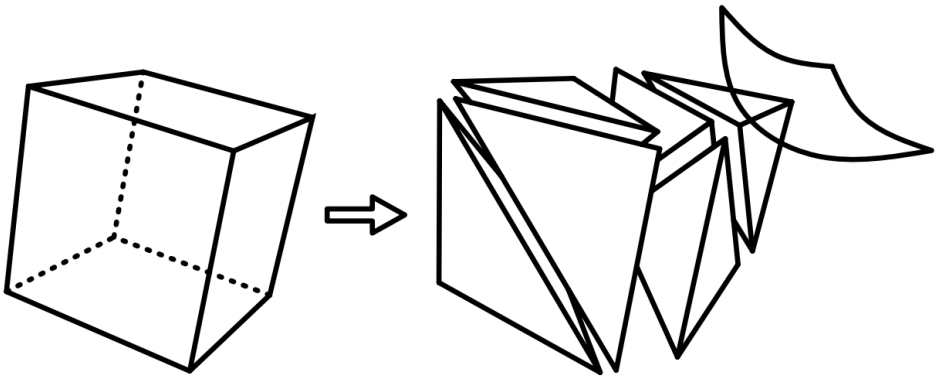
4.1 Bloomenthalova metoda poligonizacije

Bloomenthalova metoda je najbolj uporabljana poligonizacija implicitnih površin. Na začetku se poišče celico, katere rob seka površino funkcije. Ko

je najdena, se izvede poligonizacija te celice. Nato se preveri šest sosednjih celic, ki si delijo ploskev s trenutno. Za vsako celico se preveri ali je že bila obiskana, ali njeni robovi sekajo implicitno površino in ali je znotraj poligonizacijskih mej. Celice, ki ustrezajo tem pogojem, so dodane v vrsto za poligonizacijo. Nato se izbere prva iz vrste in se celoten postopek ponovi. Prednost tega pristopa je hitrost, saj se vrednosti implicitnih funkcij računa le v ogliščih celic, ki so blizu površine. Slabost tega pristopa je ta, da ne poligonizira ločenih komponent (v primeru da posamezni deli niso dovolj blizu). Pri ločenih površinah je uporabljena izčrpna poligonizacija, kjer se poligonizacijska celica premika po 3D mreži, na katero je omejeni prostor razdeljen. Postopek je časovno bolj zahteven, vendar je poligonizacija vseh ločenih površin zagotovljena.

4.2 Poligonizacija s tetraedri

Pri poligonizaciji s tetraedri se v poligonizacijski celici preverijo vrednosti oglišč. Nato se celica razdeli na šest tetraedrov (slika 4.1), vsak izmed njih ima vlogo poligonizacijske celice. Vsak rob, ki povezuje dve oglišči z nasprotnim predznakom, seka površino. Na te robove se postavijo oglišča trikotnika. Ker ima tetraeder štiri oglišča, je različnih konfiguracij šestnajst. Računanje torej poteka za šest tetraedrov in vsak ima lahko eno izmed šestnajstih kombinacij.



Slika 4.1: Tetraedrična dekompozicija celic [7].

Poligonizacija s tetraedri ima to pomanjkljivost, da se pri ostrejših prehodih površine pojavijo nazobčane strukture [7]. V takih primerih je potrebno zmanjšati velikost poligonizacijske celice pri poligonizaciji robov, kar zahteva težavno in časovno potratno implementacijo.

4.3 Poligonizacija s kockami

Koncept poligonizacije s kockami je preprost. Neposredno se polgionizira osnovna celica (ki ima obliko kocke). Glede na vrednosti oglišč se poišče presečišča robov s površino implicitne funkcije in se nato izbere eno izmed 256 konfiguracij.

Za poligonizacijo je uporabljen algoritem Marching cubes, ki je že bil implementiran v programu NeckVeins [20] za neposredno poligonizacijo volumna. Algoritem je preurejen tako, da namesto vrednosti vokslov izračuna vrednosti implicitnih funkcij in namesto interpolacije za iskanje presečišča uporablja bisekcijo. Poligonizacija s kockami pri enaki ločljivosti poteka hitreje kot poligonizacija s tetraedri in ustvari manjše število trikotnikov. To je velika prednost pred tetraedri, saj to pomeni, da se lahko v enakem času izvajanja uporabi višja ločljivost poligonizacijske mreže, kar pripomore h kvalitetnejšemu izrisu rezultata ter poligonizaciji manjših struktur, ki sicer ne

bi bile zajete. Poleg višje učinkovitosti, poligonizacija s kockami ne povzroča anomalij pri ostrih robovih, kot se to dogaja pri tetraedrih. Zaradi teh razlogov je pri večini rekonstrukcij uporabljena poligonizacija s kockami.

4.4 Implementacija in paralelizacija

Poligonizacija je implementirana v dveh stopnjah. V prvi stopnji se izračunajo vrednosti v 3D mreži, ki je določena s poljubno ločljivostjo. V drugi stopnji so vrednosti iz prejšnje stopnje uporabljene kot vrednosti oglišč poligonizacijskih celic. Zaradi deljenih oglišč sosednjih celic se tako izogne večkratnemu računanju istih vrednosti. Glede na te vrednosti se izbere poligonizacijsko konfiguracijo celice in se z določeno natančnostjo izračunajo presečišča s pomočjo bisekcije.

Za računanje vrednosti implicitne funkcije je uporabljeno osmiško drevo, ki je ustvarjeno pri algoritmu MPUI. Po drevesu se išče vse liste, katerih razdalja med središčem celice in dano točko ne presega podpornega polmera. V teh listih se vrednost funkcije izračuna in uteži.

Izčrpna poligonizacija je zaradi neodvisnosti računanj v vsaki izmed celic mreže primerna za paralelizacijo. Na procesorju sta implementirani paralelizacija postopkov poligonizacije s kockami in tetraedri, medtem ko je na grafični kartici paraleliziran le postopek poligonizacije s kockami. Paralelizaciji na procesorju sta tako kot paralelizacija delitve drevesa (poglavje 3.7) realizirani s pomočjo javanskega vmesnika Callable. Bistvena razlika je ta, da tukaj število paralelnih procesov ni omejeno na osem, temveč je omejeno z ločljivostjo volumna (po osi z).

Za paralelizacijo na grafični kartici je uporabljena knjižnica OpenCL. Program, ki se izvaja na grafični kartici uporablja že implementirane MC konfiguracije [20]. Velika težava poligonizacije rezultata MPUI je njegova drevesna struktura. Grafične kartice slabo podpirajo kompleksnejše strukture, zato je celotno drevo spremenjeno v več polj. Polja vsebujejo lokalne funkcije, središče, podporni polmer in indekse otrok posameznih vozlišč. Kljub nizko-

nivojski implementaciji drevesa se pojavljajo težave in nezmožnost izvedbe programa pri večjih globinah. Algoritem lahko pri manjši ločljivosti in manjši globini drevesa uspešno poligonizira globalno implicitno funkcijo, vendar z neopazno pohitritvijo. Vzrok za to je prevelika kompleksnost računanja in uporaba velikega števila pogojnih stavkov. Večina testiranj je zato potekala s paralelizacijo na procesorju.

Poglavje 5

Analiza in primerjava rezultatov

Primerjava je opravljena na dveh implementiranih algoritmih, MPUI in MC. Pri testiranju so bistvenega pomena parametri, ki neposredno vplivajo na rezultat. MC je v vseh testiranjih (razen pri evalvaciji natančnosti rezultatov) uporabljen v kombinaciji z Gaussovim glajenjem. Pri MPUI na rezultat vpliva več parametrov. α oziroma velikost podpornega polmera vpliva na gladkost površine, hkrati pa tudi poveča časovno kompleksnost algoritma. ε_0 ali uporabniško določena napaka določa globino delitve. Manjša kot je uporabniško določena napaka, bolj je model podoben rezultatu neposredne poligonziacije z MC. Na rezultat vpliva tudi izbira utežne funkcije, vendar njen vpliv v tem delu ni obravnavan. MPUI ni omejen z ločljivostjo volumna, zato je možna izbira poljubne velikosti poligonizacijske celice oziroma ločljivosti rezultata.

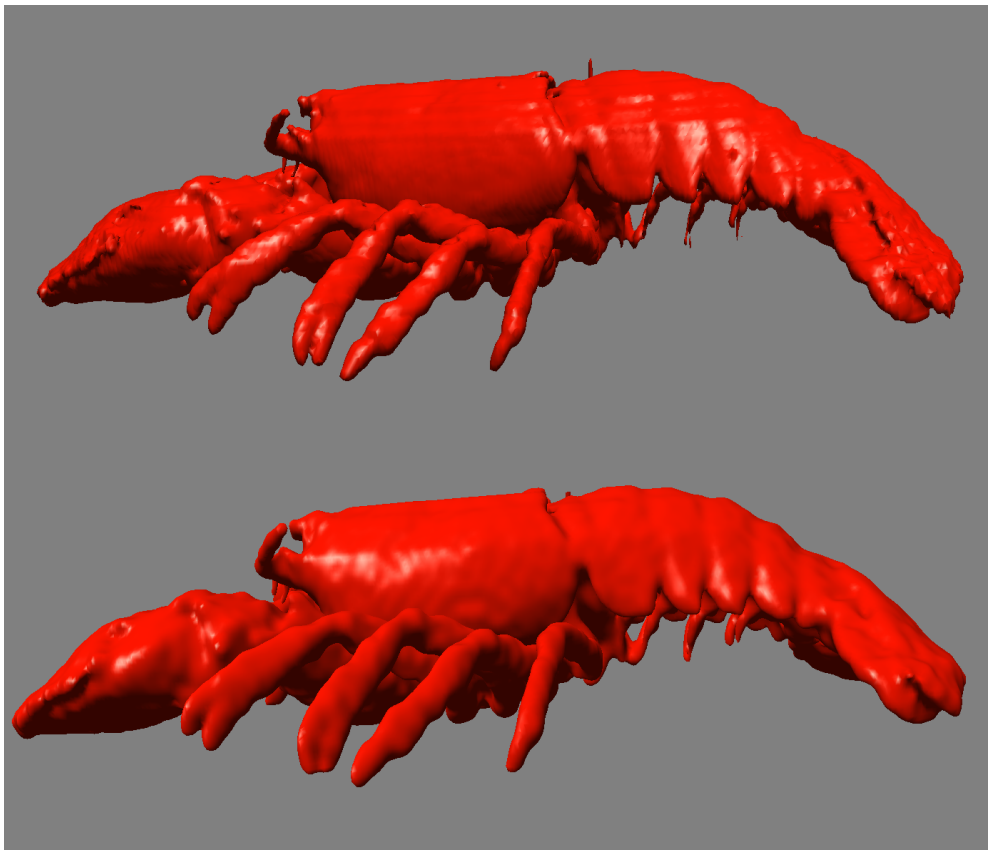
5.1 Primerjava rezultatov na različnih volumnih

Testiranja so potekala na več volumnih žil in volumnu jastoga [3]. Prva opazna razlika med obema algoritmoma je čas izvajanja. Medtem ko se zapore-

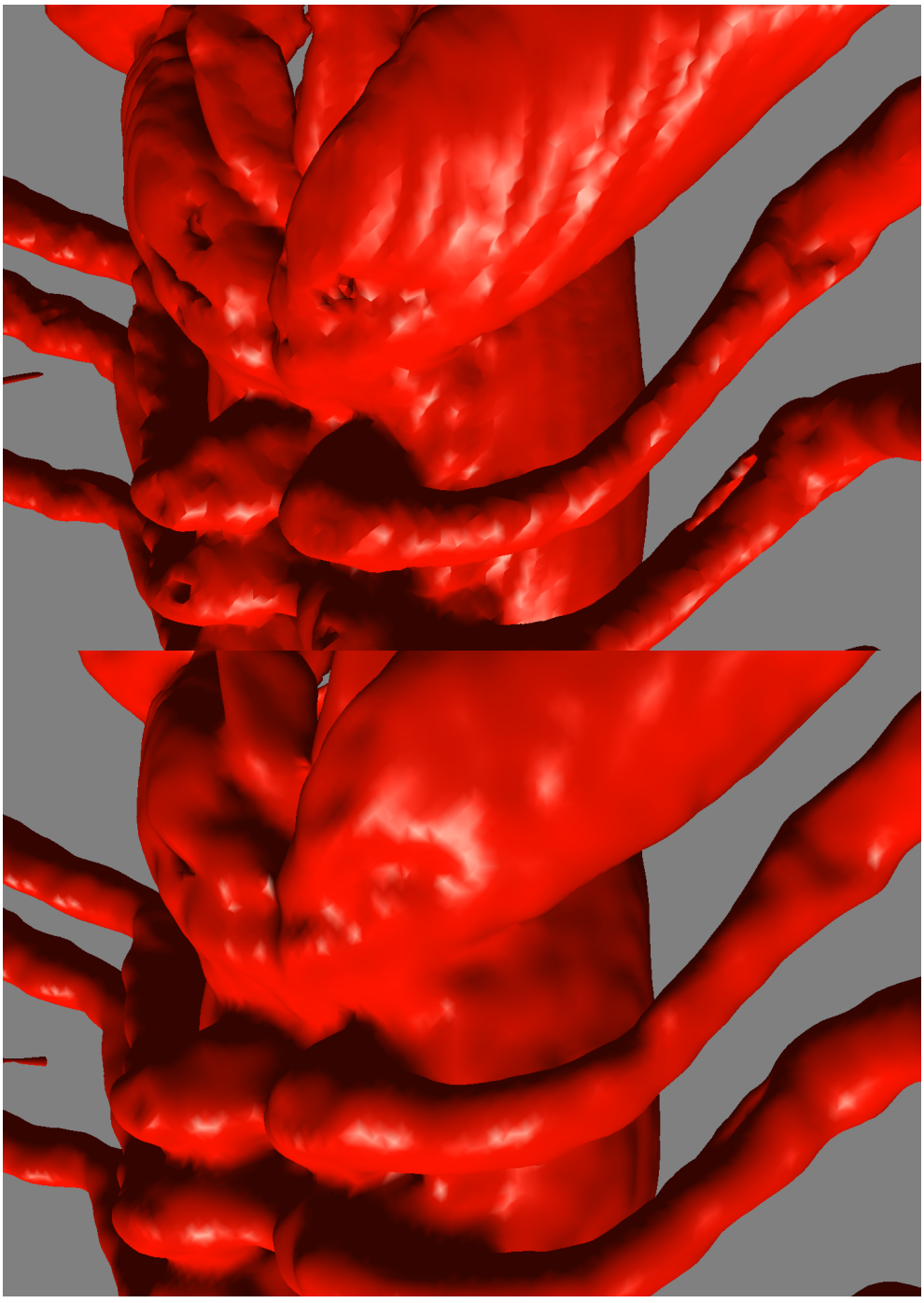
dna implementacija MC z Gaussovo eliminacijo izvede v nekaj sekundah pri manjših volumnih do pol minute pri večjih volumnih (testirano na procesorju Intel i3 2.1GHz), je MPUI kljub paralelizaciji veliko počasnejši. Odvisno od parametrov lahko rekonstrukcija z MPUI traja različno dolgo. Čas izvajanja je predvsem odvisen od kompleksnosti površine. Najbolj časovno zahteven korak je poligonizacija, saj je poleg računanj vrednosti vseh funkcij v mreži (število računanj v mreži je enako produktu vseh treh dimenzij) potrebno izračunati tudi vrednosti bisekcije pri robovih ki sekajo mrežo (ponavadi med 5 in 10 računanj za posamezen rob).

V zameno za počasno izvajanje MPUI ponuja svobodo pri manipulaciji parametrov, ki na različne načine vplivajo na rezultat. Pri ustreznih parametrih so lahko rezultati gladkejši, različne anomalije pridobljene med procesom zajema slik so lahko odpravljene. Tudi ločljivost je lahko izboljšana. Algoritma sta testirana na volumnih žil (slika 5.3), vendar so omenjene prednosti bolj vidne pri rekonstrukciji volumna jastoga (slika 5.1). Na sliki 5.2 je pri rekonstrukciji z MC vidna stopničasta površina. Njeno izrazitost do določene mere zgladi uporaba Gaussovega glajenja, vendar lahko ta pri uporabi večje konstante glajenja popači izgled modela. Vidne so tudi luknje v površini, ki jih povzroči nenatančnost samega zajema slik. Obe anomaliji sta v veliki meri odpravljeni z uporabo algoritma MPUI.

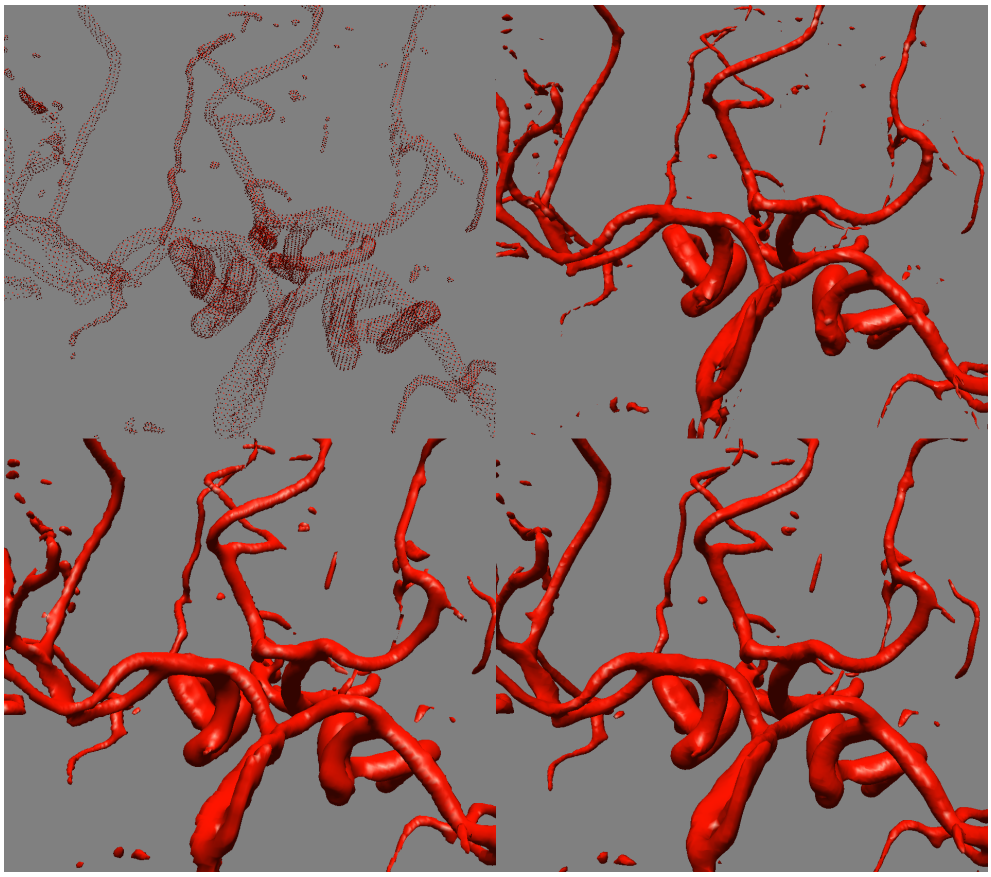
Kljub vsem dobrim lastnostim, ima MPUI pri rekonstrukciji tudi večjo pomanjkljivost. Veliko število ločenih vokslov, ki se nahajajo zelo blizu se pri ustvaritvi oblaka točk pretvori v množico različno orientiranih točk na zelo majhnem prostoru. Aproksimacija površine je v takih primerih težavna in lahko nastanejo kroglaste anomalije. To težavo se da odpraviti z dodatnim vzorčenjem točk pri majhnih strukturah v koraku ustvaritve oblaka točk. V tem delu omenjena rešitev ni implementirana.



Slika 5.1: Rekonstrukcija 3D modela jastoga iz volumna. Na zgornji sliki je volumen rekonstruiran z Gaussovim glajenjem in algoritmom Marching cubes, spodaj pa z MPUI. Za Gaussovo glajenje je uporabljen parameter $\sigma = 0.5$. MPUI uporablja parametre $\alpha = 2$, $\varepsilon = 0.008$ in velikost poligonizacijske celice 0.003.



Slika 5.2: Rekonstrukcija 3D modela iz volumna jastoga. Na zgornji sliki je volumen rekonstruiran z Gaussovim glajenjem in algoritmom Marching cubes, spodaj pa z MPUI. Parametri so enaki kot na sliki 5.1.



Slika 5.3: primerjava rekonstrukcije 3D modela možganskih žil, levo zgoraj je oblak točk, nato si v smeri urinega kazalca sledijo algoritmi MC, MPUI s tetraedrično poligonizacijo ter MPUI z MC poligonizacijo.

5.2 Evalvacija natančnosti rekonstrukcij

Določanje natančnosti posameznih rekonstrukcij poteka v dveh korakih. V prvem so na izbranem volumnu testirani vsi algoritmi. Rezultati teh algoritmov so poligonizirani modeli. V drugem koraku je na vseh poligoniziranih modelih izvedena vokselizacija površine. Za vokselizacijo je uporabljen program [13], ki poligonizirano površino postavi v 3D mrežo in kot rezultat vrne vse celice iz mreže, ki sekajo površino modela.

Cilj primerjave je določiti natančnost postopkov, katerih namen je vzpostavitev ravnotežja med natančnostjo in izgledom modela. Ker algoritem MC interpolira vrednosti volumna pri poligonizaciji, je uporabljen kot kriterij za določanje natančnosti ostalih pristopov. Ostali pristopi so Gaussovo glajenje volumna z MC poligonizacijo in MPUI z različnimi velikostmi podpornega polmera.

Za najboljše ravnovesje med natančnostjo in izgledom modela je pri Gaussovem glajenju s poligonizacijo MC izbran koeficient glajenja $\sigma = 0.5$. Pri manjših vrednostih je model preveč robat, pri prevelikih pa je natančnost rezultata prenizka (slika 5.4). Pri MPUI algoritmu sta izbrana modela z velikostjo podpornega polmera $\alpha = 1.5$ in $\alpha = 2$.

Primerjava diskretiziranih površin je opravljena s pomočjo programa Matlab. Rezultat vokselizacije je mreža velikosti n^3. Vrednosti vokslov v mreži, ki sekajo površino modela, so nastavljeni na logično enico, ostale vrednosti vokslov pa na ničlo. Vsa ujemanja med površinami se izračunajo z logično operacijo “and” nad obema matrikama. Rezultat je 3D matrika z logičnimi enicami na mestih kjer so vrednosti istoležnih vokslov v obeh matrikah enaki ena. Število ujemanj je seštevek vseh vrednosti matrike.. Ker je poleg ujemanj površin pomemben tudi položaj oziroma odmik zgrešitev, se v naslednjem koraku površina vseh algoritmov (razen osnovnega MC, s katerim se ostali primerjajo) poveča. Vsi voksli, ki imajo vsaj enega soseda z logično vrednostjo ena, so nastavljeni na enico. Ujemanja so izračunana na enak način. Rezultat je število vokslov v mreži, ki so za največ eno stranico voksla oddaljeni od površine s katero jih primerjamo.

Rezultati primerjav so prikazani v dveh tabelah. Tabela 5.1 prikazuje rezultate diskretizacije površine v mrežo velikosti 100x100x100, tabela 5.2 pa v mrežo velikosti 200x200x200. Prva vrstica tabel predstavlja število vseh vokslov, ki vsebujejo vrednost ena in predstavljajo površino. Druga vrstica predstavlja število vokslov (z logično vrednostjo 1), ki se po položaju v mreži ujemajo z voksli površine algoritma MC brez glajenja. Naslednja vrstica predstavlja delež števila ujemanj v številu vseh vokslov površine pri algoritmu MC brez glajenja. Četrta vrstica, označena z * predstavlja število ujemanj, pri čemer se za ujemanje šteje tudi, če sta voksla diskretizirane površine na sosednjih mestih (razdalja med površinama približno enaka stranici enega voksla). Zadnja vrstica predstavlja delež ujemanj četrte vrstice v številu vseh vokslov algoritma MC.

V mreži velikosti 100x100x100 ima MC z Gaussovim glajenjem približno 12 odstotnih točk več ujemanj kot oba MPUI modela. Kljub temu je število ujemanj pri obeh relativno visoko (med 70% in 84%). Pri naslednji primerjavi se lahko razbere porazdelitev odklona površin in največji odklon. Testirano je, koliko ujemanj imata površini, če je dovoljen odklon enega voksla. Rezultati kažejo da se pri MPUI modelih celotna površina ujema (tabela 5.1), kar pomeni, da je največja možna razdalja med površinama enaka stranici enega voksla (dolžini 0.01). Pri MC z Gaussovim glajenjem je ujemanj 99.35%, kar pomeni da obstaja 0.65% površine, ki je za več kot en voksel oddaljena od površine osnovnega modela. Odklon pri površini MPUI je bolj enakomerno porazdeljen. To prednost bi se dalo izkoristiti za doseganje še večje natančnosti z manipulacijo različnih parametrov. V tem primeru bi prav prišla zelo majhna sprememba odmika implicitnih funkcij (slika 5.4).

Pri mreži z dvakrat večjo stranico se delež ujemanj pri vseh treh algoritmih zmanjša. Pri MC z Gaussovim glajenjem je število ujemanj za 18 odstotnih točk manjše, pri MPUI pa za približno 25. Število ujemanj s toleranco enega voksla je pri vseh treh algoritmih med 97% in 98% (slika 5.6). Glede na rezultate je očitno, da razlika v velikosti podpornega polmera (med 1.5 in 2, podobno bi verjetno veljalo tudi za 0.75) nima bistvene vloge pri

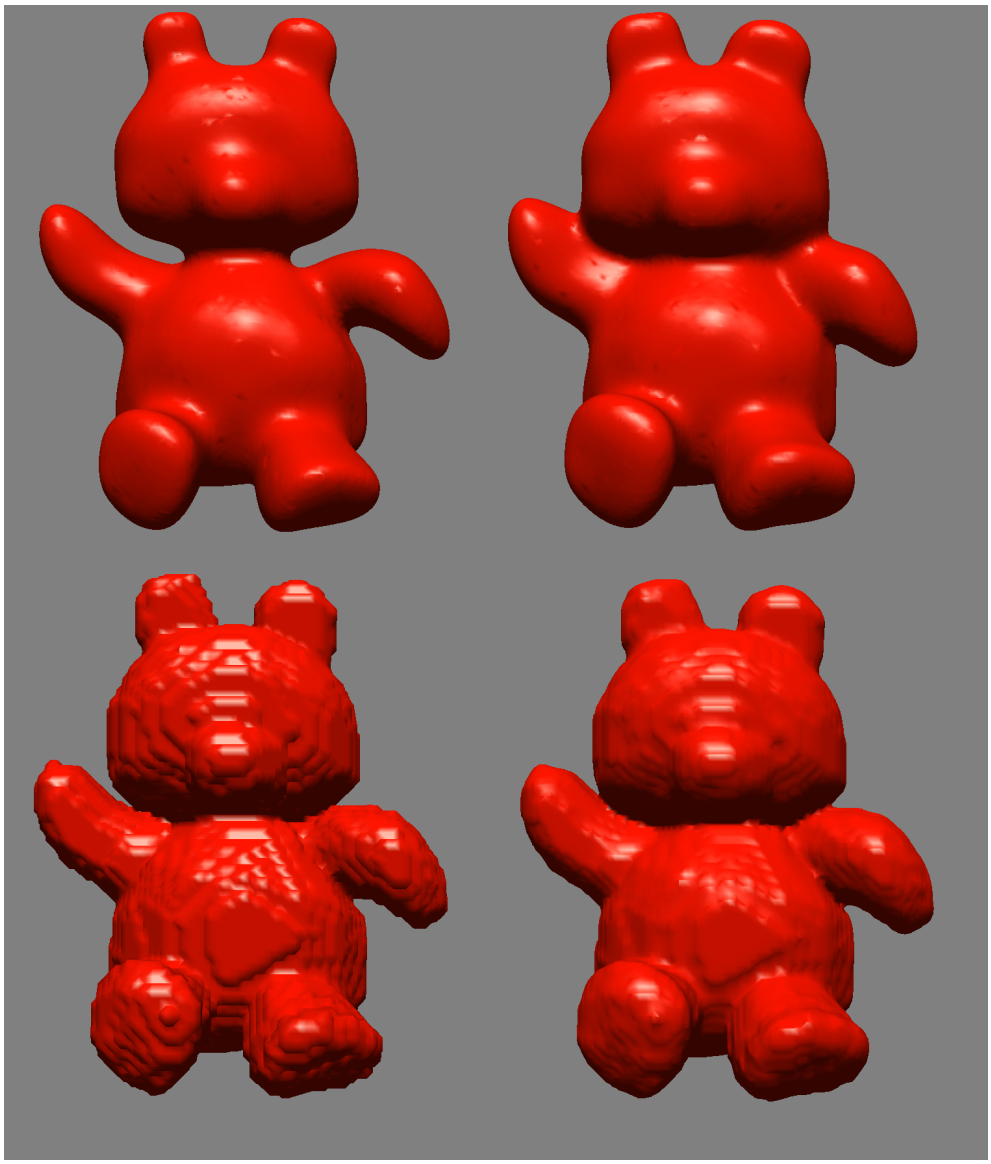
Algoritem	MC	MC + GG $\sigma = 0.5$	MPUI $\alpha = 2$	MPUI $\alpha = 1.5$
Št. vokslov	9386	9295	9002	9049
Št. ujemanj	/	7719	6652	6685
Delež ujemanj	/	83.04%	70.87%	71.22%
Št. ujemanj*	/	9325	9386	9386
Delež ujemanj*	/	99.35%	100%	100%

Tabela 5.1: Primerjava natančnosti površin različnih modelov v mreži velikosti $100x100x100$.

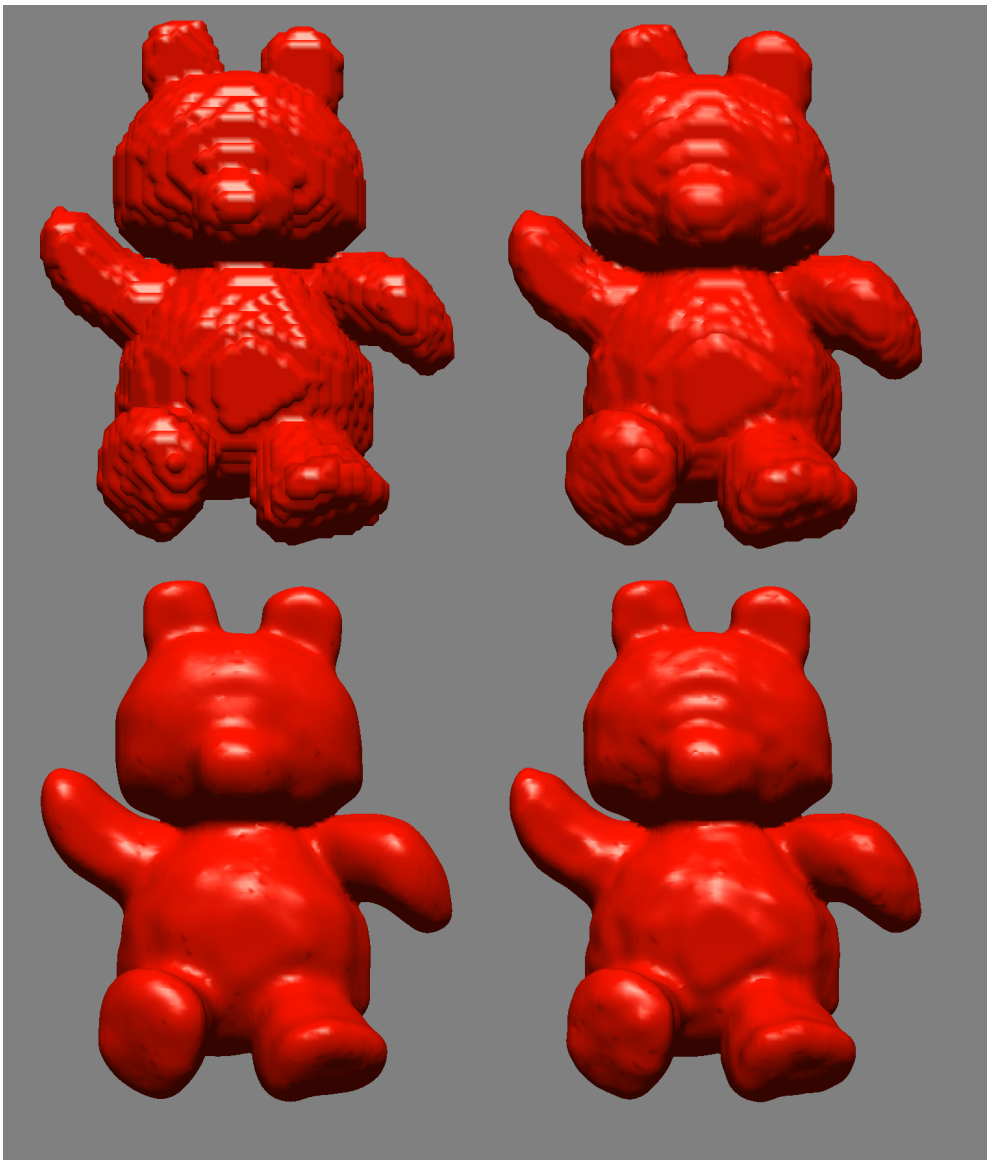
Algoritem	MC	MC + GG $\sigma = 0.5$	MPUI $\alpha = 2$	MPUI $\alpha = 1.5$
Št. vokslov	37440	37252	36012	36146
Št. ujemanj	/	24426	17288	17528
Delež ujemanj	/	65.24%	46.18%	46.82%
Št. ujemanj*	/	36545	36495	36402
Delež ujemanj*	/	97.61%	97.48%	97.23%

Tabela 5.2: Primerjava natančnosti površin različnih modelov v mreži velikosti $200x200x200$.

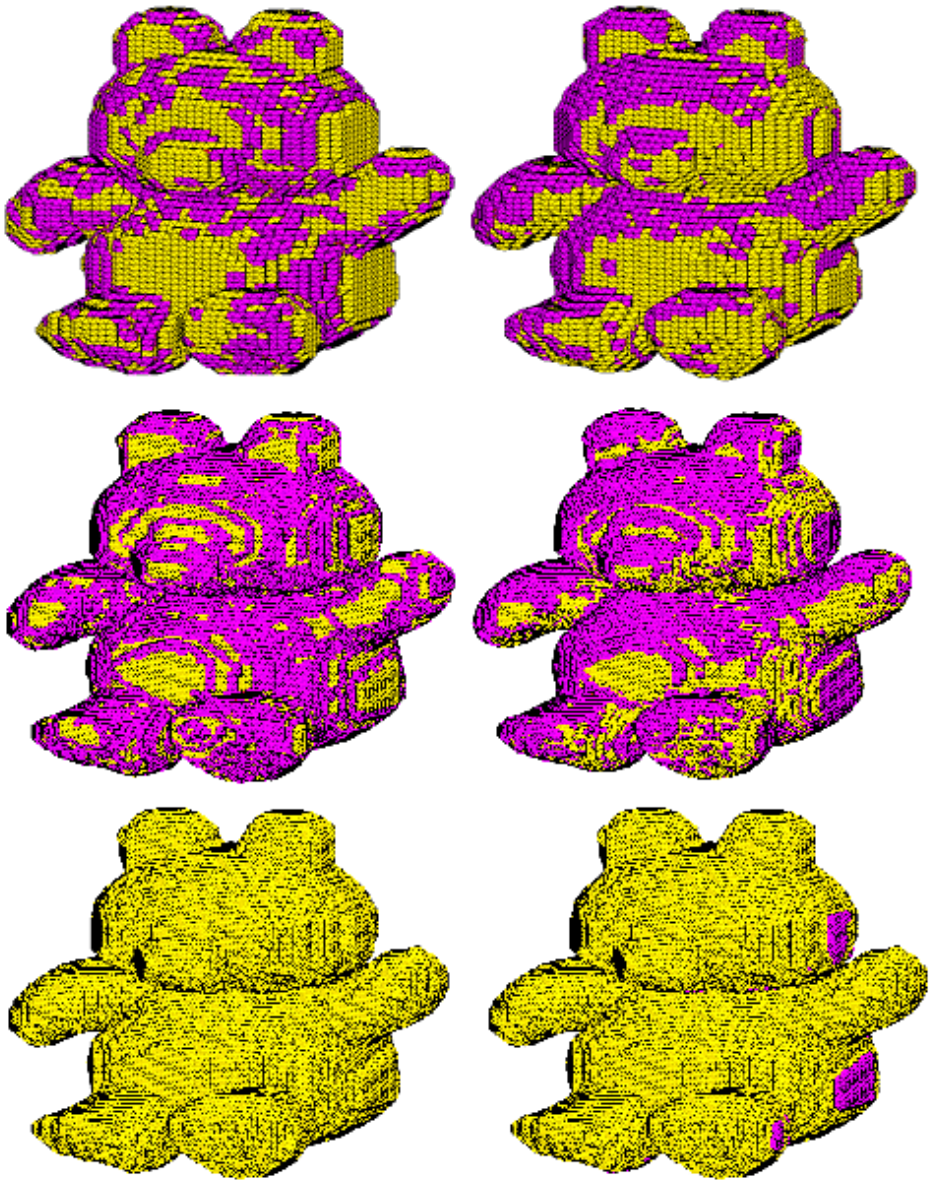
natančnosti rekonstrukcije, vpliva pa na gladkost rezultata (slika 5.5). Kljub temu, da rezultati kažejo na bolj natančno rekonstrukcijo MC z Gaussovim glajenjem, ima MPUI poleg zadovoljive natančnosti še veliko bolj naraven izgled rekonstruiranega modela (slika 5.5). Model je bolj gladek in nima težav z zlivanjem bližnjih struktur. Z Gaussovim glajenjem take gladkosti brez deformacij modela ni mogoče doseči, kar ponazarja slika 5.4. Pri večjem koeficientu glajenja pride do zlivanja površin glave in trupa modela, ki se vizualno vedno bolj oddaljuje od osnovnega modela.



Slika 5.4: primerjava rekonstrukcije 3D modela medveda [2], levo zgoraj je rekonstruiran z MPUI in pozitivnim odmikom (0.006) nato si v smeri urinega kazalca sledijo MPUI z negativnim odmikom (-0.006), algoritem MC z Gaussovim glajenjem ter konstanto glajenja $\sigma = 0.6$ in MC z Gaussovim glajenjem s konstanto $\sigma = 0.4$. Odmik pri MPUI je konstanta, ki se prišteje vrednosti izračunane lokalne funkcije. Z negativnim odmikom se tako poveča površina modela, s pozitivnim pa zmanjša (negativne vrednosti so znotraj modela, pozitivne zunaj).



Slika 5.5: primerjava rekonstrukcije površine medveda [2], levo zgoraj je rekonstruiran z MC nato si v smeri urinega kazalca sledijo algoritmi MC z Gaussovim glajenjem, MPUI s parametrom $\alpha = 2$ ter MPUI s parametrom $\alpha = 1.5$.



Slika 5.6: zgrešitve pri primerjavi rekonstrukcije z MC (levo, z Gaussovim glajenjem, $\sigma = 0.5$) in MPUI (desno, s podpornim polmerom $\alpha = 2$). V prvi vrsti sta modela vokselizirana z ločljivostjo 100x100x100, v drugih dveh pa z 200x200x200. V zadnji vrsti se kot zgrešitve obravnavajo voksli, ki so za več kot en voksel oddaljeni od originalne površine. Rumeni voksli prikazujejo ujemanja rekonstrukcij teh dveh algoritmov z osnovnim MC, vijolični pa zgrešitve. Večina zgrešitev se nahaja na območjih, kjer se stopničasta površina algoritma MC ne ujema z gladko površino glajenja oziroma aproksimacij MPUI.

Poglavje 6

Sklepne ugotovitve

V diplomskem delu je implementiran algoritem MPUI za rekonstrukcijo 3D modelov iz oblaka točk. Vhodni podatki za MPUI so pridobljeni z ustvaritvijo oblaka točk iz volumetričnih podatkov. Množica uteženih lokalnih funkcij, ki je rezultat MPUI algoritma, je poligonizirana z uporabo izčrpne poligonizacije. Glede na konfiguracije poligonizacijske celice sta uporabljeni poligonizacija s tetraedri in s kockami. Zaradi počasne izvedbe MPUI in poligonizacije sta algoritma paralelizirana za vzporedno izvajanje na procesorjih.

Z izboljšavo obravnave manjših struktur pri aproksimaciji ali ustvaritvi oblaka točk in pohitritvijo poligonizacije ima algoritem potencial za uporabo pri medicinski diagnozi. Adaptivna aproksimacija, s katero se lahko kljub pomanjkljivostim volumetričnih podatkov vizualno približamo realnemu izgledu objekta, omogoča široko možnost uporabe algoritma. Pristop je primeren za večino rekonstrukcij, kjer so vhodni podatki orientirane točke ali volumni. Algoritem se že uporablja za rekonstrukcijo modelov iz 3D laserskih skeniranj, vendar bi lahko imel tudi večjo vlogo pri rekonstrukciji medicinskih podatkov. Zaradi gladkosti in realnega izgleda modelov so lahko te uporabljeni za učne namene. Možnost izbire visoke ločljivosti modela lahko pride prav pri 3D tiskanju objekta, kjer visoka ločljivost pripomore k natančnejšemu končnemu izdelku.

Za potrebe medicinske diagnostike bi bila smiselna implementacija spro-

tnega prilagajanja parametrov, glede na natančnost površin. Natančnost bi lahko bila izračunana kot število ujemanj med diskretizirano površino in volumnom. Implicitna funkcija je lahko neposredno diskretizirana, zato se lahko natančnost preverja pred poligonizacijo, s čimer se bistveno skrajša čas izvedbe. S tem bi lahko dosegli večjo vlogo implicitne rekonstrukcije površin pri diagnostiki s pomočjo medicinskih volumnov.

Literatura

[1] Java callable documentation. http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Callable.html.

[2] Teddybear model. http://groups.csail.mit.edu/graphics/classes/6.837/F03/models/.

[3] Volvis real medical datasets. http://volvis.org/, 2005.

[4] Java matrix benchmark. https://code.google.com/p/java-matrix-benchmark/wiki/RuntimeCorei7v2600_2013_10, 2013.

[5] Simon Žagar. Vizualizacija žil tilnika z OpenGL-om. Diplomsko delo, 2012.

[6] Nina Amenta, Sunghee Choi, in Ravi Krishna Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry*, 19(2):127–153, 2001.

[7] Jules Bloomenthal. An Implicit Surface Polygonizer.

[8] Jonathan C. Carr, Richard K. Beatson, Jon B. Cherrie, Tim J. Mitchell, W. Richard Fright, Bruce C. McCallum, in Tim R. Evans. Reconstruction and representation of 3D objects with radial basis functions. Objavljeno v *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, strani 67–76. ACM, 2001.

[9] F. Distance approximations for rasterizing implicit curves. *ACM Transactions on Graphics (TOG)*, 13(1):3–42, 1994.

[10] Sarah F. Gibson. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. Objavljeno v William M. Wells, Alan Colchester, in Scott Delp, editors, *Medical Image Computing and Computer-Assisted Interventation — MICCAI'98*, del 1496 of *Lecture Notes in Computer Science*, strani 888–898. Springer Berlin Heidelberg, 1998.

[11] John C Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.

[12] Michael Kazhdan, Matthew Bolitho, in Hugues Hoppe. Poisson surface reconstruction. Objavljeno v *Proceedings of the fourth Eurographics symposium on Geometry processing*, 2006.

[13] Dirk-Jan Kroon. Polygon2Voxel. `http://www.mathworks.com/matlabcentral/fileexchange/24086-polygon2voxel`, 2009.

[14] Peter Lancaster in Kes Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of computation*, 37(155):141–158, 1981.

[15] Žiga Lesar. Vizualizacija medicinskih volumetričnih podatkov v realnem času. Diplomsko delo, 2014.

[16] Marc Levoy. Display of surfaces from volume data. *Computer Graphics and Applications, IEEE*, 8(3):29–37, May 1988.

[17] William E. Lorensen in Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, Avgust 1987.

[18] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, in Hans-Peter Seidel. Multi-level Partition of Unity Implicits. Objavljeno v *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[19] Christian Schumann, Mathias Neugebauer, Ragnar Bade, Bernhard Preim, in Heinz-Otto Peitgen. Implicit vessel surface reconstruction for visualization and CFD simulation. *International Journal of Computer Assisted Radiology and Surgery*, 2(5):275–286, 2008.

[20] Anže Sodja. Segmentacija prostorskih medicinskih podatkov na GPE. Diplomsko delo, 2012.